UNIDAD 4

DESARROLLO BÁSICO DE APLICACIONES ANDROID

1. Conceptos básicos sobre el desarrollo Android
2. Estructura de un proyecto Android
3. Introducción al IDE Android Studio
4. Layouts
5. Atributos
6. Nuestros primeros experimentos en Java

1 - Conceptos Básicos sobre el desarrollo Android

Imagen de Giphy.com

Android es un sistema operativo de código abierto, que existe sobre un núcleo linux modificado, sobre el que se monta el resto del sistema.

Las aplicaciones si que funcionan de una forma muy parecida a las aplicaciones linux. A cada una, el sistema le da un identificador de usuario único. Es decir: cada app se comporta como un usuario independiente. Se hace así para asignar a cada app permisos solo sobre sus archivos, de forma que no puedan tocar nada del sistema ni de otras aplicaciones. Este es el principio del menor privilegio, y se hace para garantizar la seguridad en el sistema.

Además, se ejecutan en un Entorno Virtual propio, por el que no pueden ser afectados por otras aplicaciones a no ser que tengan permisos. Lo veremos más adelante.

Algunas palabras clave sobre el desarrollo en Android son:

- Actividad: Elemento básico de visualización (“pantalla”). Crea interfaz de usuario. Las actividades trabajan por separado para lograr un objetivo común. Hereda de la clase Activity.

- Servicio: Ejecutan procesos en segundo plano, no tienen una interfaz de usuario. Pueden usarse para comunicación entre actividades o entre aplicaciones. Hereda de la clase Service.

- APK : Paquete en el que se compila una aplicación Android, y actúa de instalador.

- AAB : App Bundle. Sustituto del APK en la APP Store a partir de 2021/2022. Es un conjunto de apks modulares, que no contienen toda la app, sino que cada apk contiene un trozo, de forma que en vez de descargar toda la apk al instalar, el aab crea una apk personalizada con esos trozos según tus necesidades (región, arquitectura,idioma...)

- SDK: Software Development Kit, trae librerías y herramientas necesarias para desarrollar un programa en Android.

- Android Monitor (DDMS): Debugger de Android. Imprescindible.

- Android Debug Bridge (ADB): Herramienta que permite comunicar el entorno de desarrollo (y Android Monitor) con un dispositivo o emulador, para usarlo para depurar. Cada fabricante facilita el suyo, aunque hay algunos genéricos.

- Android Virtual Devices (AVD): Dispositivo virtual de android que proporciona Android Studio, emulador.

- Platform Tools: Herramientas que dan soporte a las distintas plataformas y versiones de Android, incluye entre otras cosas el ADB.

- Manifest: Archivo de configuración, donde se deben declarar los componentes de la aplicación, entre otras cosas como permisos o bibliotecas. Está escrito en XML.

- Proyecto es la entidad superior, que engloba a uno o varios módulos.

- Módulo es un conjunto de archivos de origen y configuraciones de compilación que divide al proyecto en unidades que se pueden compilar, probar y depurar independientemente.

- Vista. No es lo mismo que la Estructura de carpetas. Nosotros normalmente estaremos en la vista Android, que se corresponde con una vista lógica del proyecto, pero no con la estructura de carpetas.

- Gradle. Es el compilador, tiene ficheros de configuración asociados en nuestro proyecto, guardan cosas como la versión del api usada.

- API Android: Application Programing interface. En el caso de Android, son unas librerías con sus funciones, que nos ofrece el SO para desarrollar aplicaciones. Cada API Android viene asociada a una versión del SO. Por ejemplo, la Api 17 es de Android 4.2 (Jelly Bean), y la api 29 es la de Android 10.

  Cada nueva API introduce y elimina funciones con respecto a las anteriores. Pero una app que use una funcionalidad específica de una API concreta (por ejemplo, 23 - Marshmallow) no puede usarse en un dispositivo como una api anterior (Por ejemplo, 19 - KitKat)

Así, si encontramos en la Play Store una app que no podemos descargar porque “No es compatible con tu dispositivo”, es porque quien la ha programado, lo ha hecho usando una API mínima posterior a la nuestra.

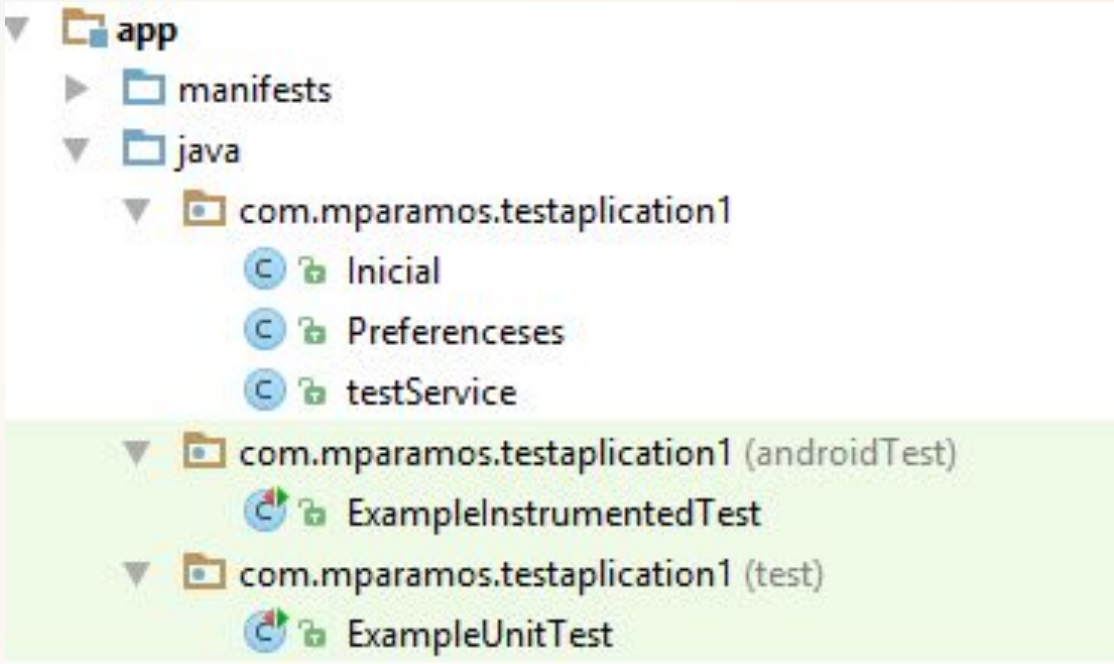
En el compilador Gradle se establece una versión mínima de la api para la app. Cuanto más alta sea, más y mejores funcionalidades tendremos disponibles. Cuanto más baja sea, más dispositivos serán compatibles con nuestra app. La decisión correcta estará en un compromiso basado en un estudio de mercado.

2 - Estructura de un proyecto Android

Imagen de Giphy.com

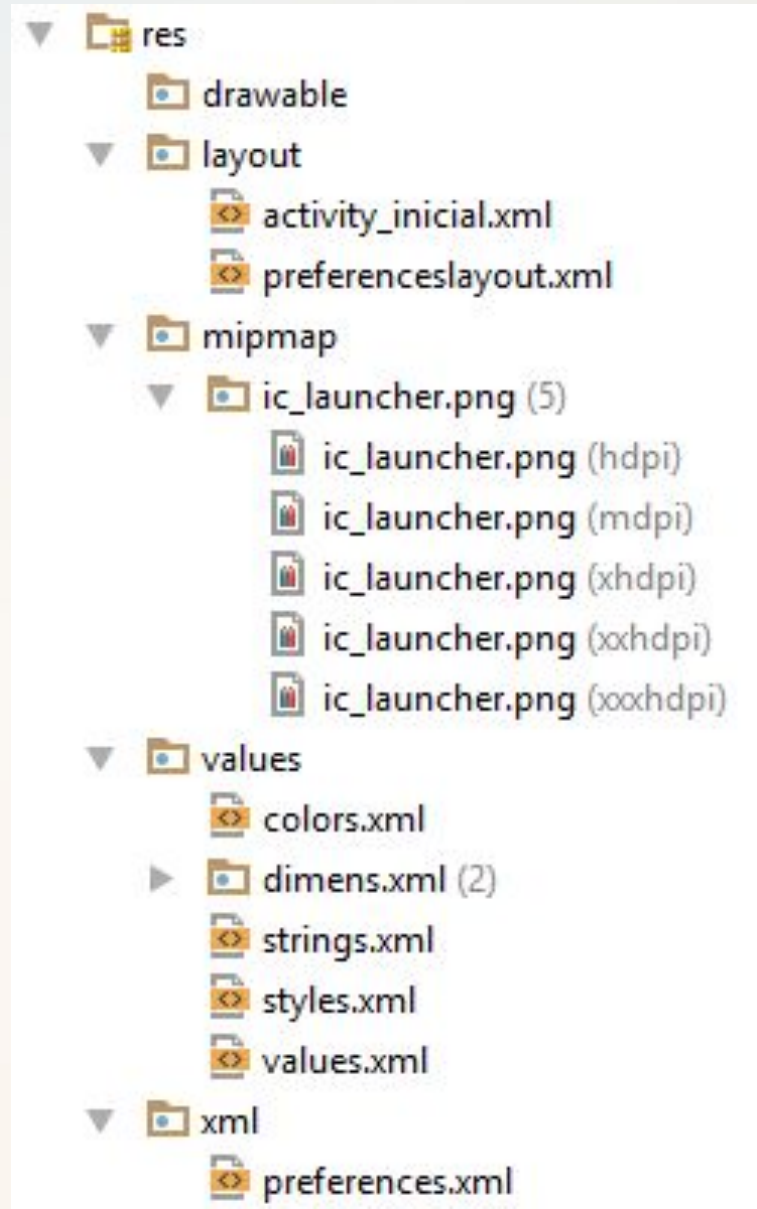
Normalmente, veremos nuestro proyecto utilizando la vista Android, ya que es la más cómoda. En ella, encontramos las siguientes carpetas:

- Carpeta Java: Contiene las clases Java que modelan el comportamiento de la app. Se puede dividir en paquetes. Se pueden incluir paquetes y clases auxiliares normales. También contiene los test, que veremos más adelante.

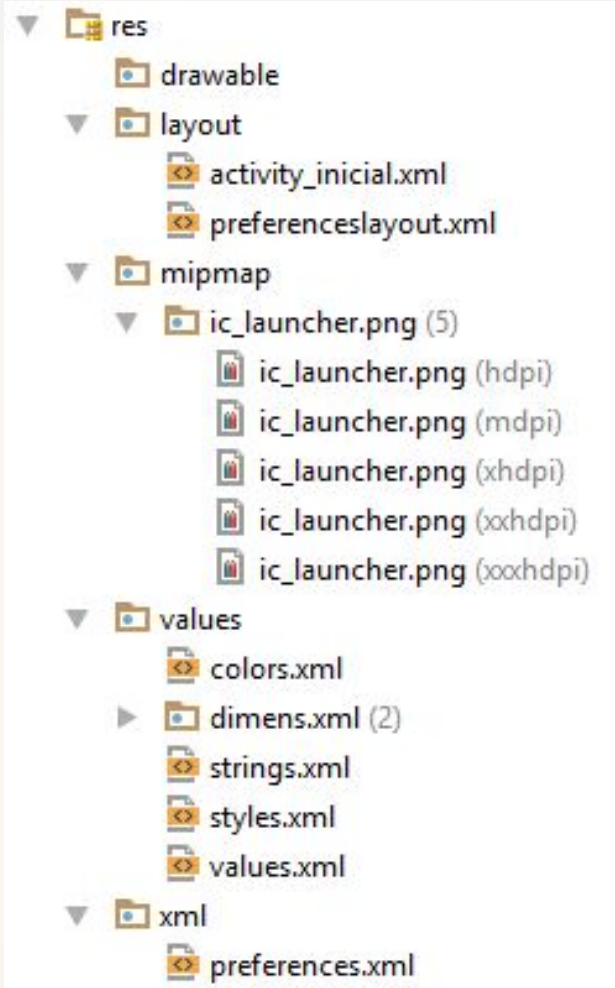


- Carpeta res: Contiene todos los recursos que no son código Java, como imágenes, sonidos, layouts…

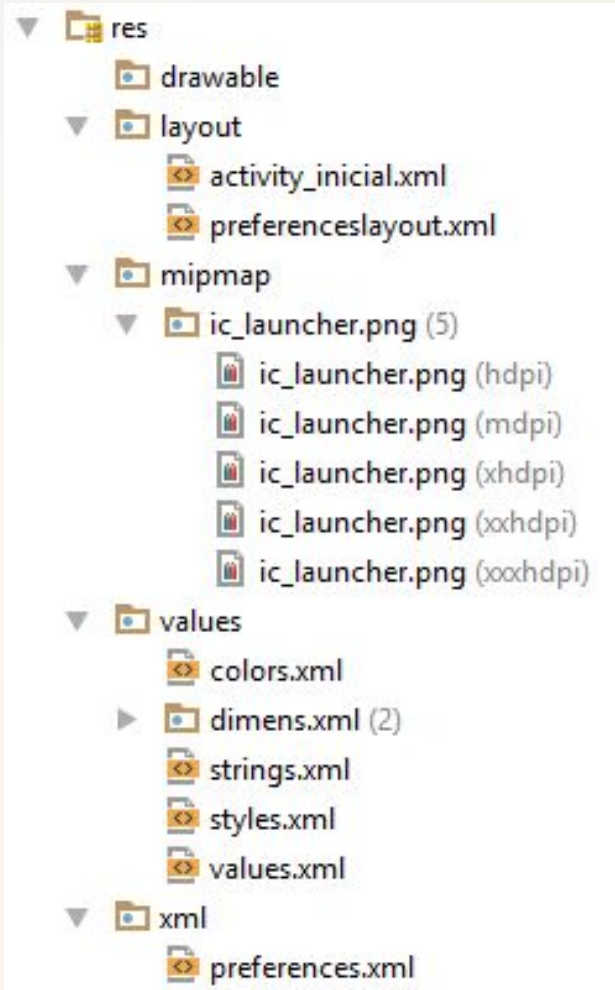
Por defecto, contiene estas subcarpetas y archivos:



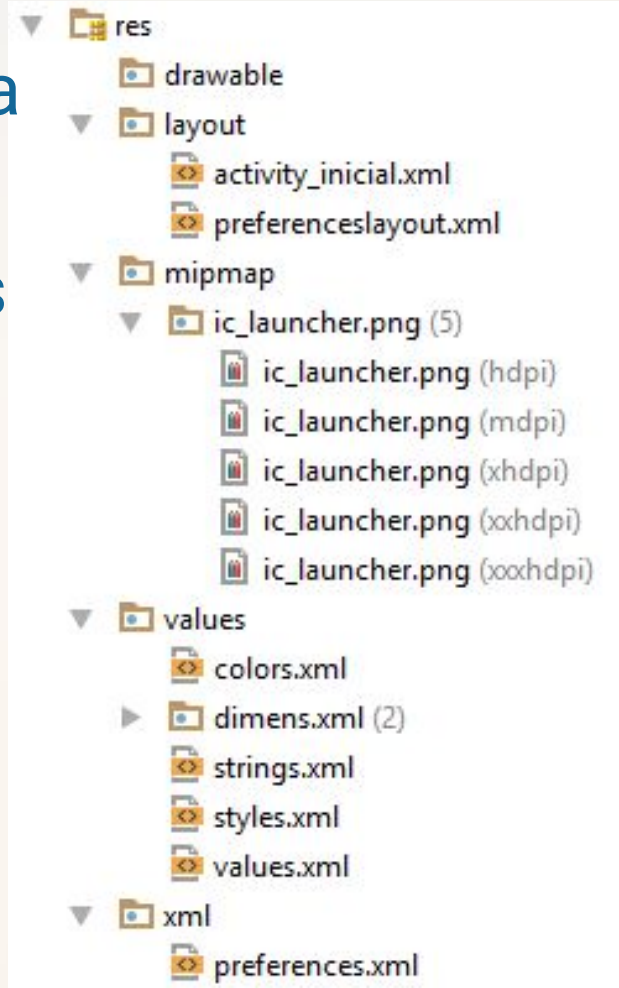
- **anim:** archivos XML que definen las animaciones.
- **drawable:** Contiene las imágenes. Pueden estar en formato png, jpg, bmp, webp, gif o heif desde android 10.
- **layout:** Los archivos layout son xml que definen la vista de cada una de nuestras “pantallas” android. Ya definiremos “pantallas” mejor más adelante.
- **xml:** Aquí hay archivos de preferencias, traducciones, y otros.



- **menu:** Son XML que definen las plantillas de nuestros menús.
- **mipmap:** Tiene los iconos de la aplicación, normalmente con varias resoluciones.
- **raw:** Contiene archivos multimedia.
- **values:** Define los valores constantes, como los colores, los estilos o los textos de la app.



- **values/strings.xml:** Define los textos que puede ver el usuario de la app.
- **values/styles.xml:** Define los estilos que se pueden usar en los elementos de una app.
- **values/color.xml:** Aquí se define la paleta de colores de la App.



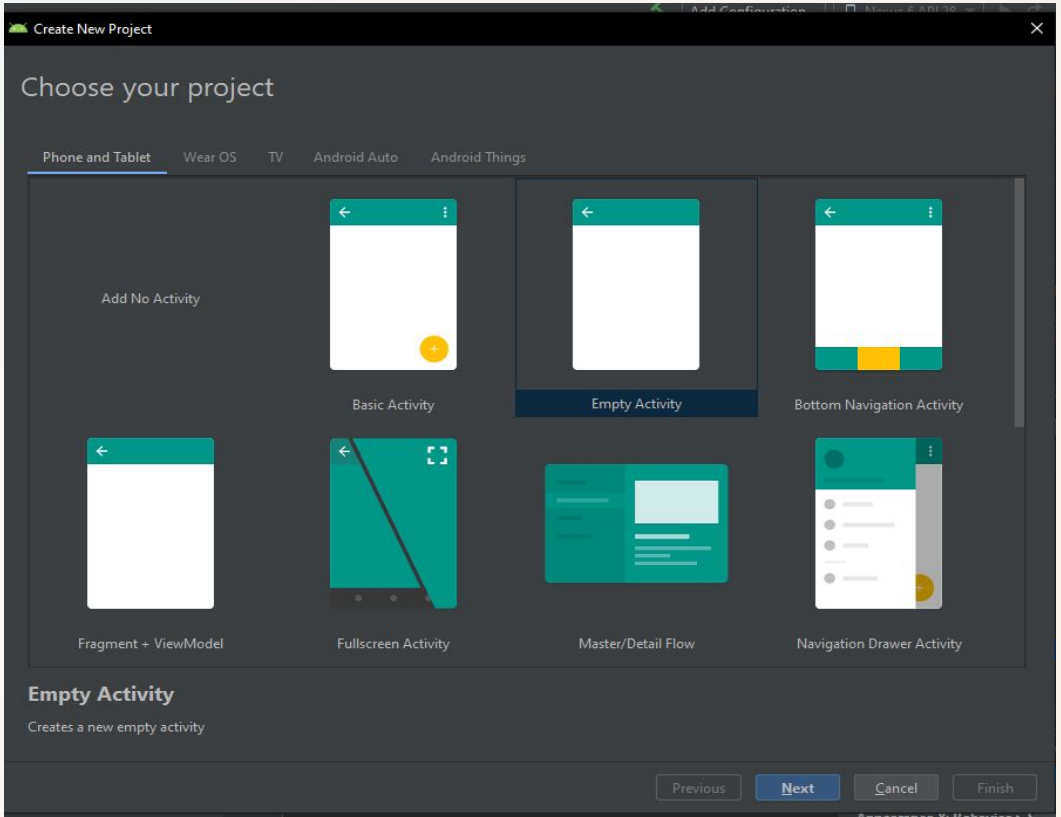
3 - Introducción al IDE Android Studio

Imagen de Giphy.com

Es el IDE Oficial para Android. Funciona en base a proyectos. Un proyecto representa a una Aplicación, que puede tener uno o varios módulos (Uno para Android, otro para Wear, otro para Auto, otro para TV...).

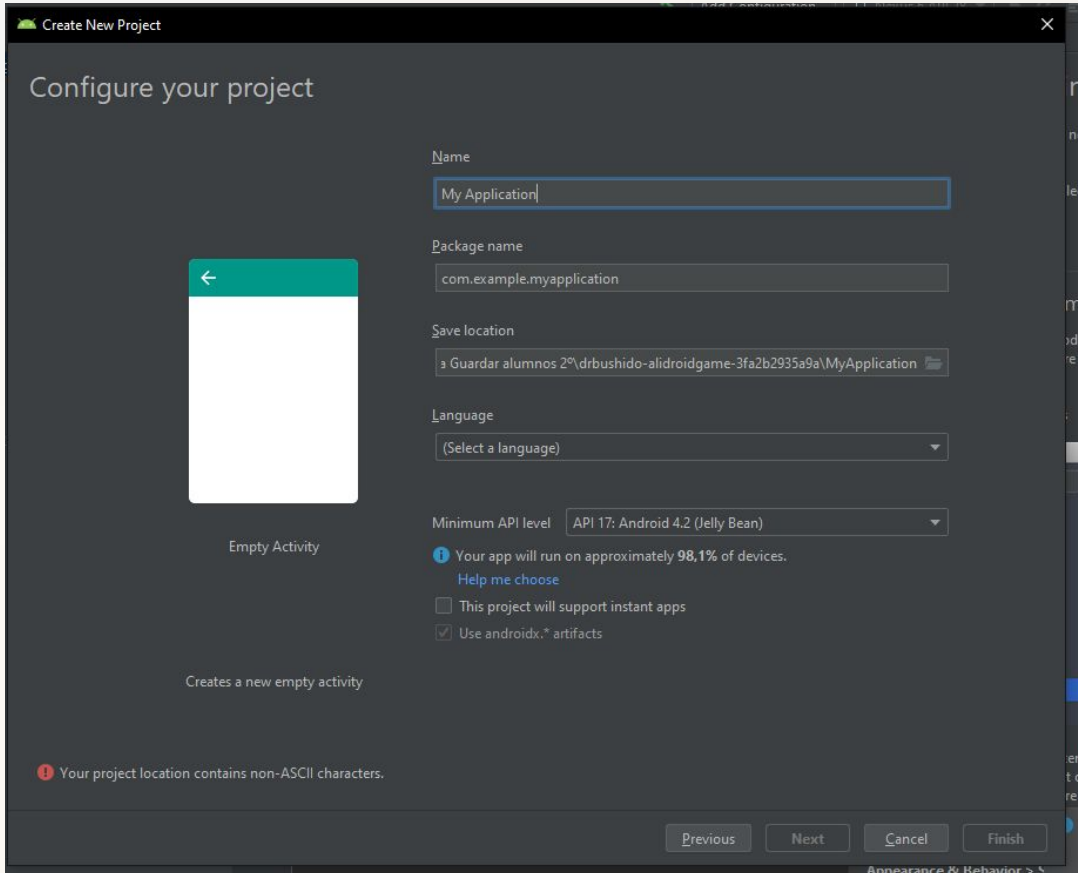
Vamos a familiarizarnos con ella creando un proyecto.

Al darle a Crear proyecto, nos aparece esta ventana. Nos permite elegir el módulo por defecto en las pestañas superiores (luego se pueden añadir más), y el diseño inicial de nuestra primera actividad. Por claridad en la explicación, vamos a elegir Empty Activity en el módulo para teléfonos y tablets.



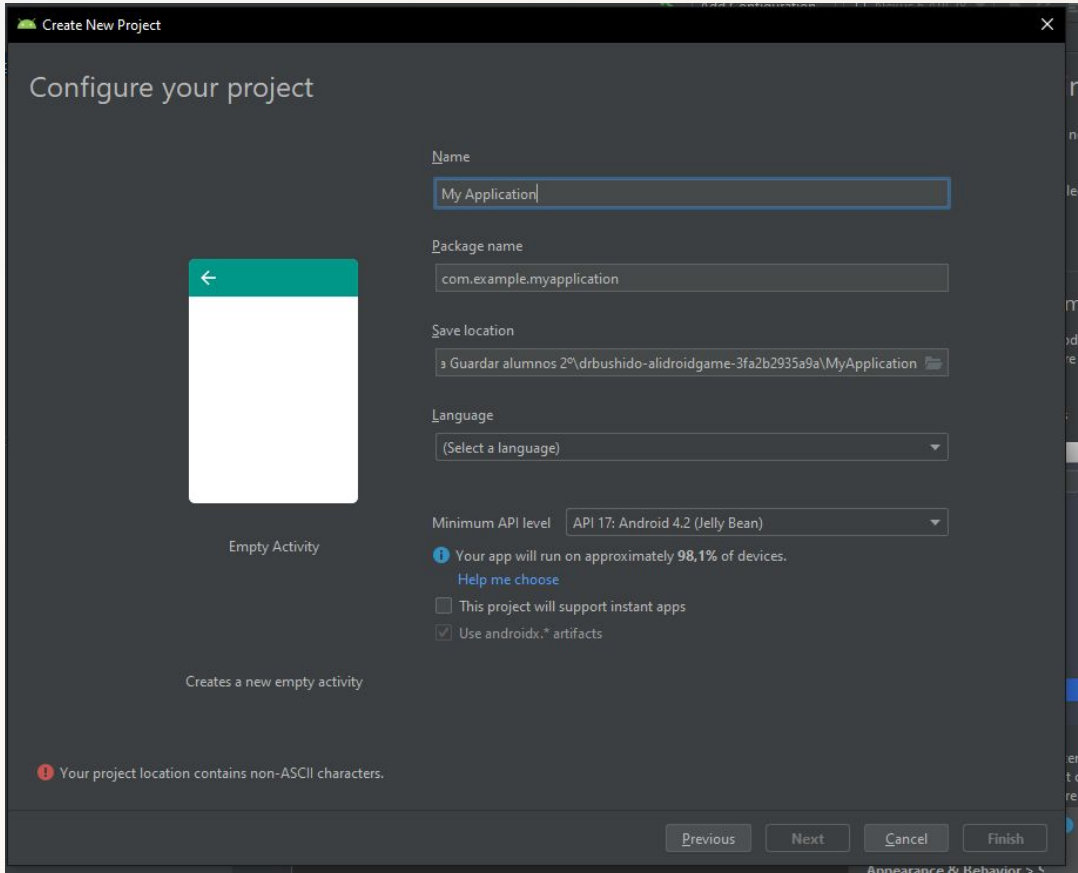
En esta pantalla vamos a elegir:

- Nombre de la app
- Paquete: se usa para identificar app y empresa. Al menos debe tener dos palabras separadas por punto. lo normal es poner: (dominioweb). empresa.proyecto
- Localización: No debe tener espacios ni caracteres especiales,acentos o ñ. **La carpeta debe estar vacía.**



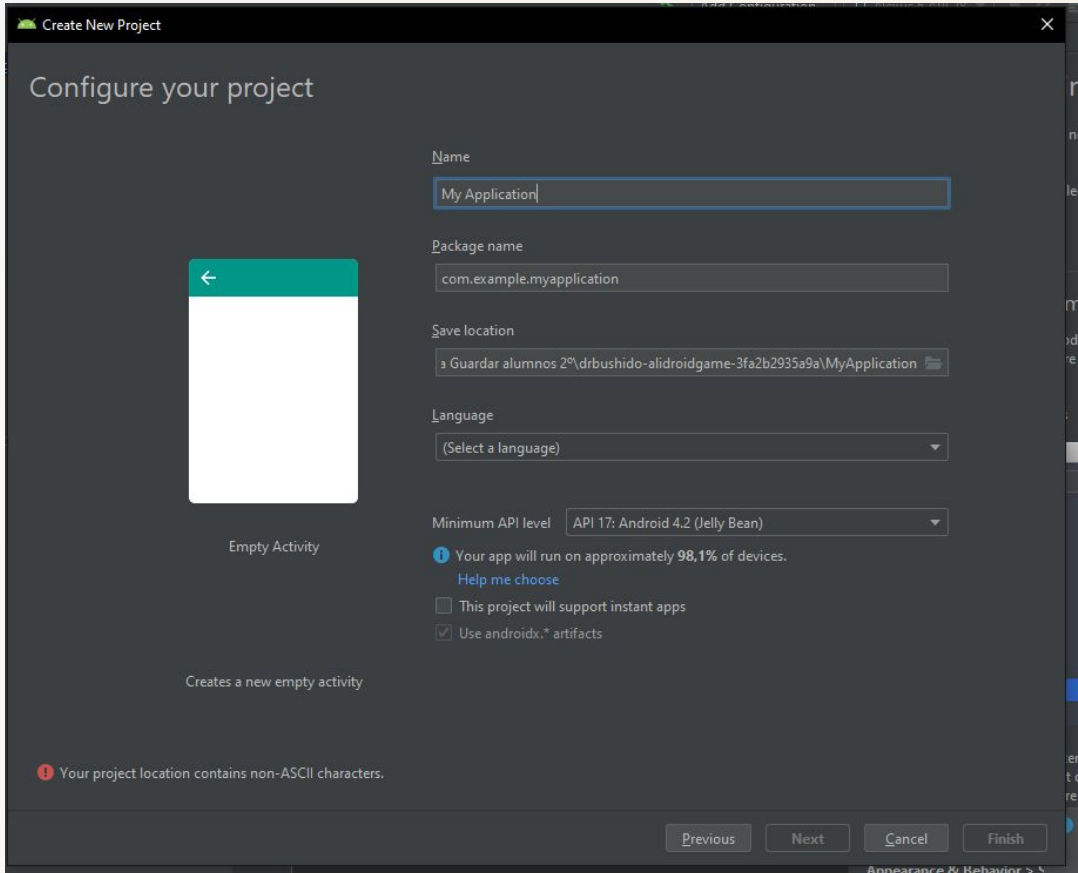
- Lenguaje: Java o Kotlin. Por ahora elegiremos Java, y usaremos Kotlin más adelante.
- Api mínima: Determina las funcionalidades disponibles para el desarrollo de la app, y cuántos usuarios van aproximadamente a poder usar la app.

  Busca el compromiso.

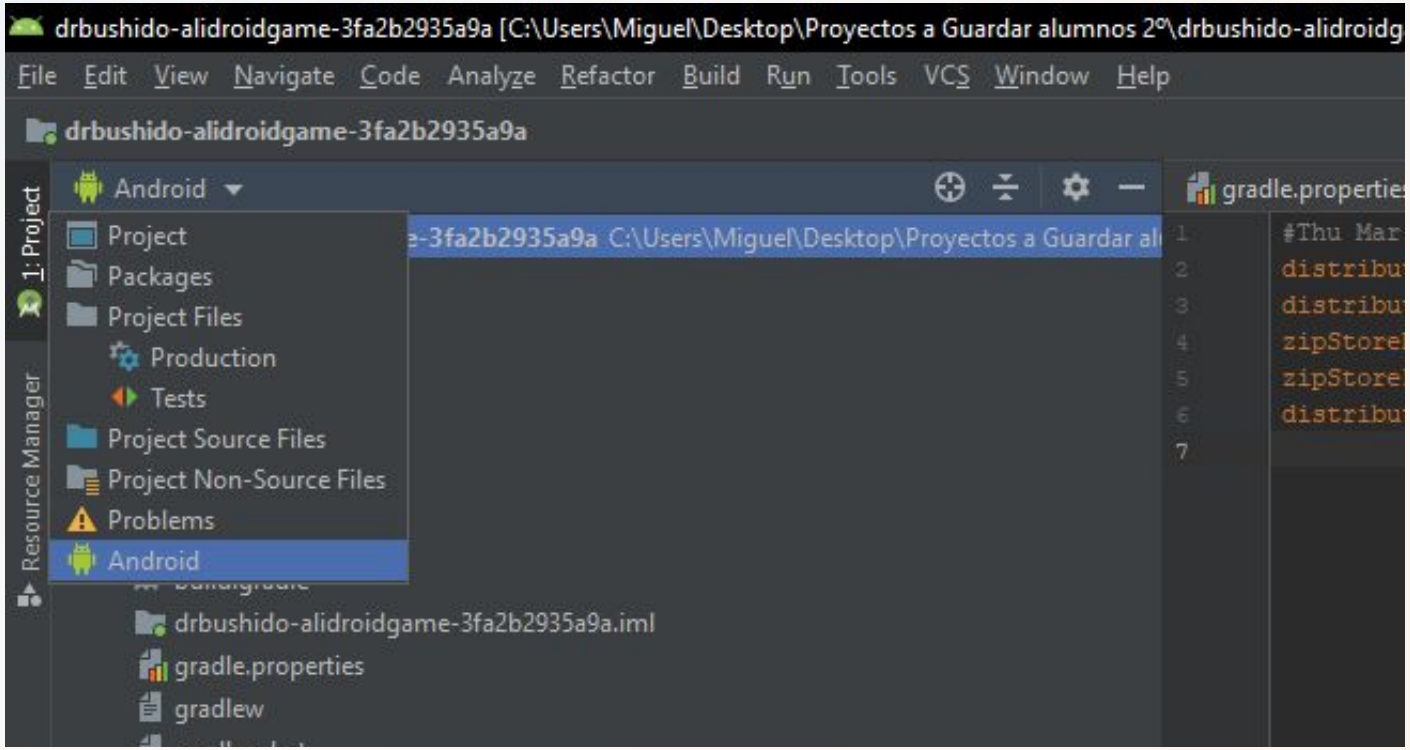


- Soporte de aplicaciones instantáneas: Si tu app soportará el uso de aplicaciones con capacidad para ejecutarse dentro de otra, o de una web, sin necesidad de instalarse en el dispositivo.
- Artefactos androidx:

  Son la nueva forma de denominar las librerías que no pertenecen al SO, sino que son auxiliares. Es recomendable activarlas para no perder funcionalidad.



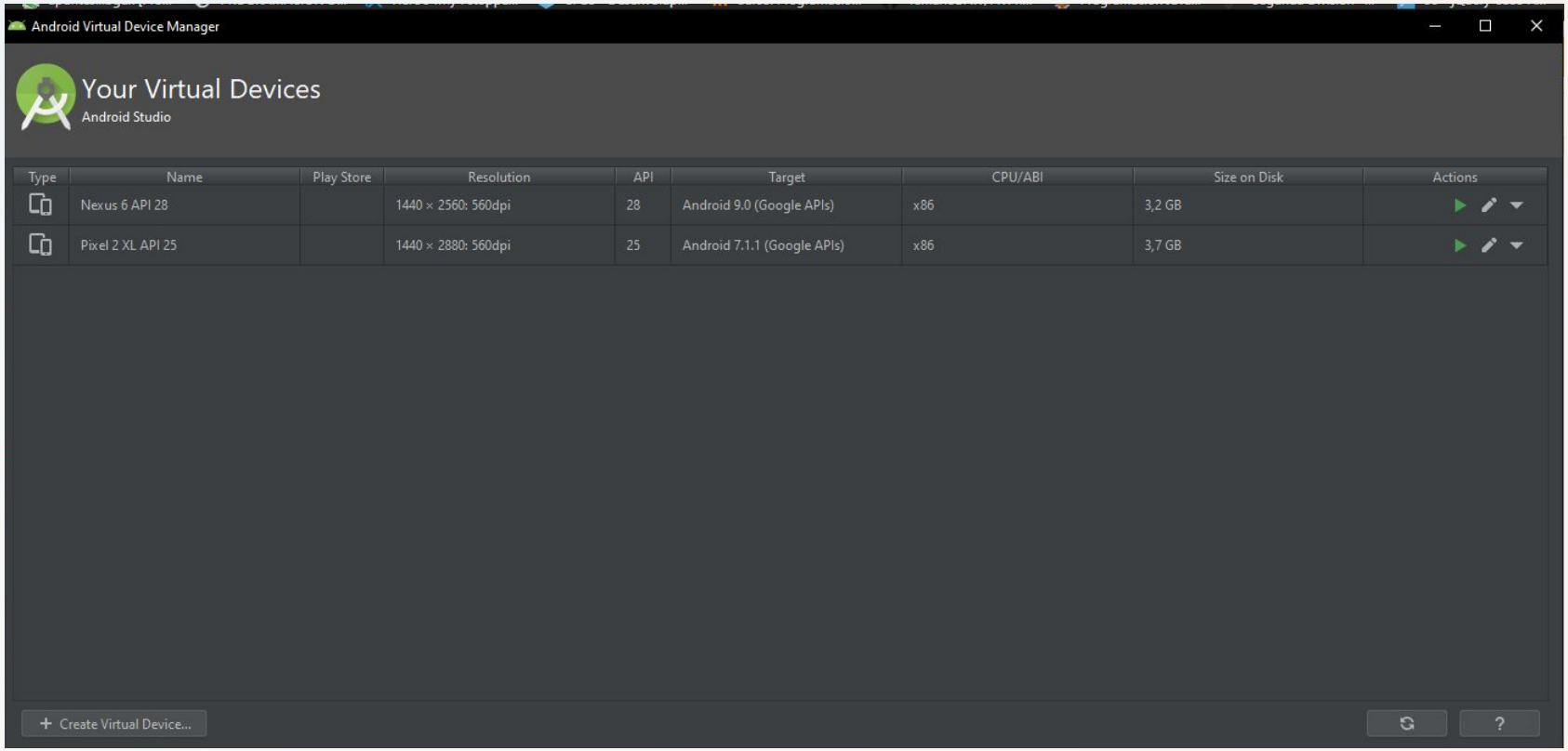
Una vez creada la app, todo nos debe resultar familiar de otros IDE. Tenemos abajo la terminal, y a la derecha podemos elegir entre las diferentes vistas.



Una cosa importante: A través de la variable de Java R, se accede a todos los recursos de la carpeta res/. Si vemos en el código Java errores porque “No encuentra” la variable R, es solo que no se ha compilado aún el código.

Normalmente, tras añadir un nuevo recurso a la carpeta res, es necesario recompilar el proyecto antes de poder usar ese recurso desde Java.

Una sección importante dentro del IDE es tools -> AVD Manager. (AVD = Android Virtual Device). Desde aquí creamos y gestionamos nuestros dispositivos Virtuales, en los que probar las aplicaciones.



Para conectar los dispositivos directamente, es necesario instalar los ADB Drivers que debe proporcionar el fabricante de vuestro dispositivo, se instalan en Windows, y si todo va bien, permiten depurar directamente en vuestro teléfono a través de Android Studio.

Imagen de Giphy.com

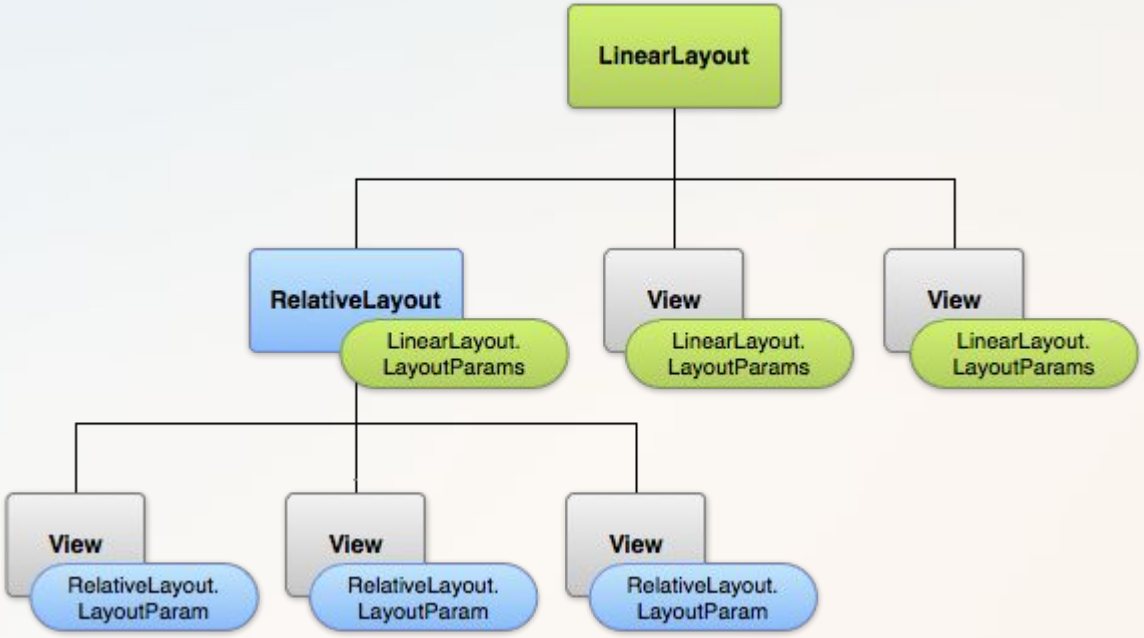
CENEC MALAGA

Dos conceptos Básicos:

- View: Elemento básico para dibujar bloques visuales. Es un área rectangular que puede contener distintos diseños. De el heredan todos los elementos visuales como Button, TextView, EditText, ImageView...

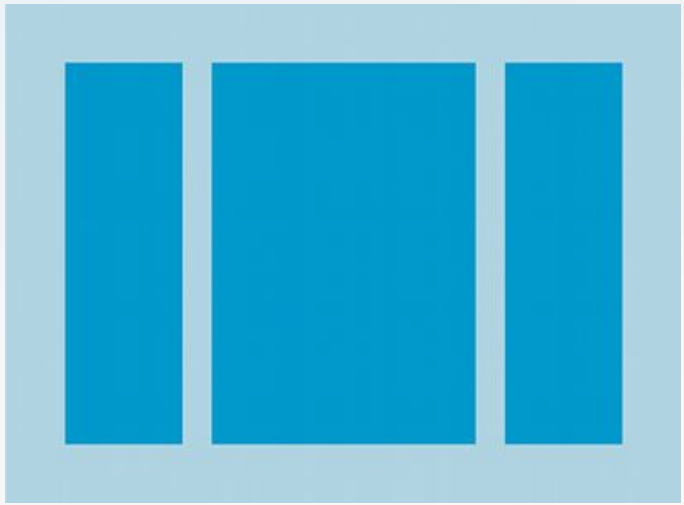
- ViewGroup: Clase especial de View que puede contener otras View u otros ViewGroup.

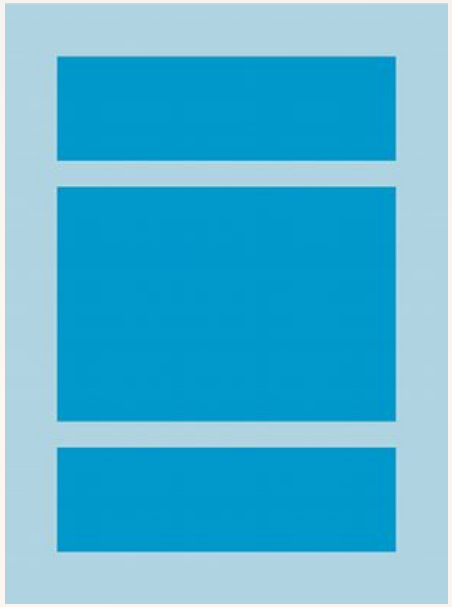
Cada ViewGroup tiene unos Parámetros de diseño en XML, con estructura layout_algo, que se pueden usar en los hijos directos de este ViewGroup, y que tienen sentido para usarse en él.



Todos los ViewGroup definen al menos layout_width y layout_height, donde se puede especificar anchura y altura relativas al layout padre.

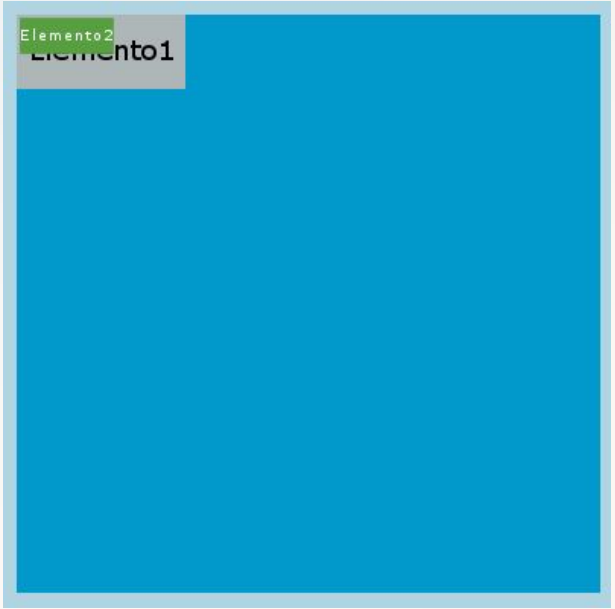
LinearLayout es un ViewGroup que alinea a todos los hijos en una única dirección, usando el atributo android:orientation. Respeta los márgenes y gravedad de cada campo interior.





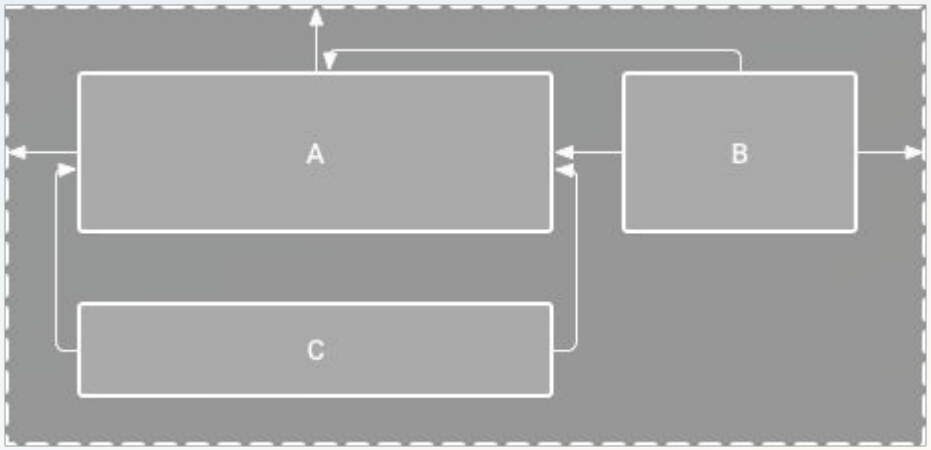
Hay dos parámetros de diseño para LynearLayout, android:layout_gravity (Para alineación) y android:layout:weight (Para hacer proporciones de espacio asignado) .

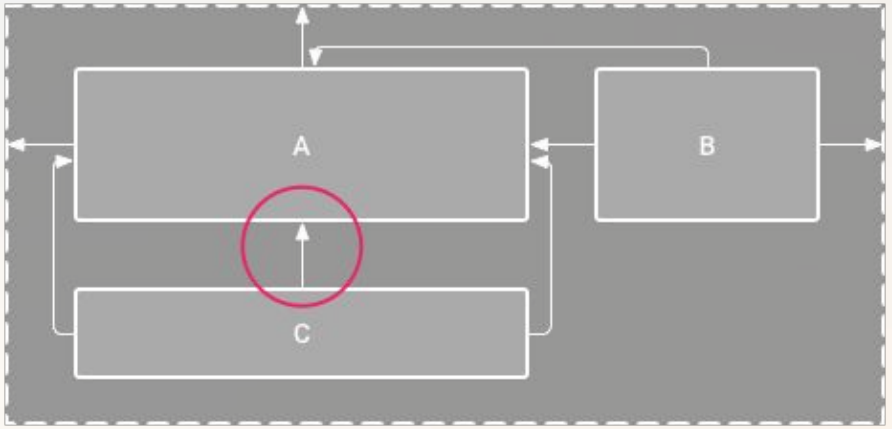
FrameLayout está pensado para bloquear una región de la pantalla, y mostrar en ella un solo ítem Se puede usar para superponer varios elementos los unos a los otros. Se alinearán todos por defecto arriba a la izquierda.



Es posible usarlo para tener múltiples elementos alineados usando el parámetro de diseño android:layout_gravity , pero no es muy recomendable, pues habrá problemas de escalado.

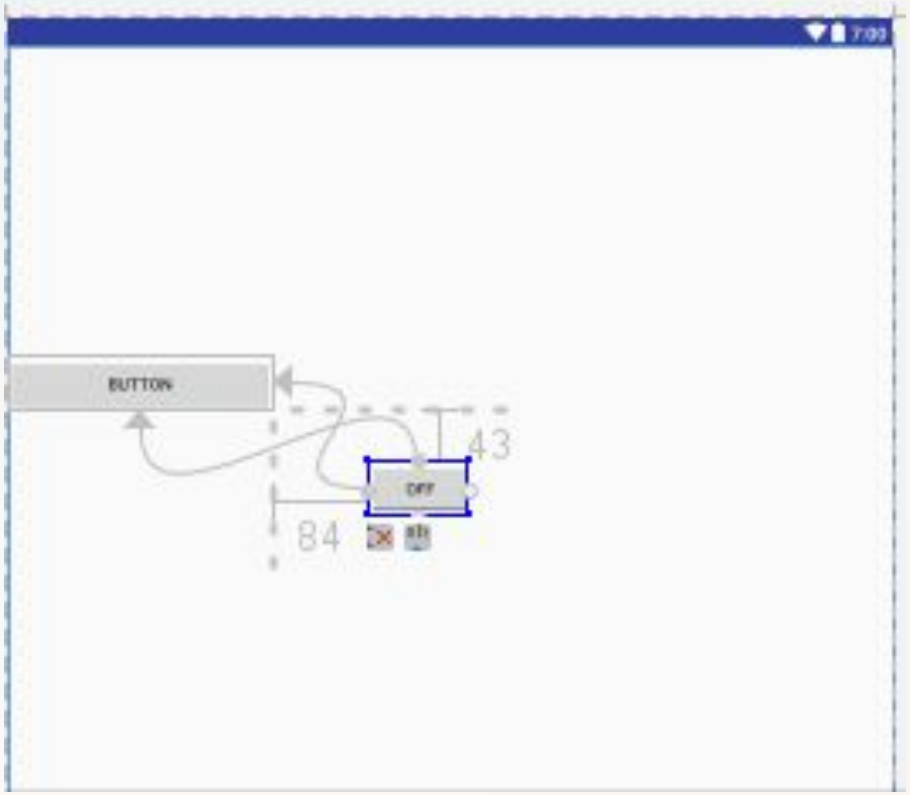
ConstraintLayout permite posicionar objetos de forma flexible, según restricciones, al menos una vertical y una horizontal. Es evolución del RelativeLayout, y permite crear diseños adaptables a distintos tamaños de pantalla.



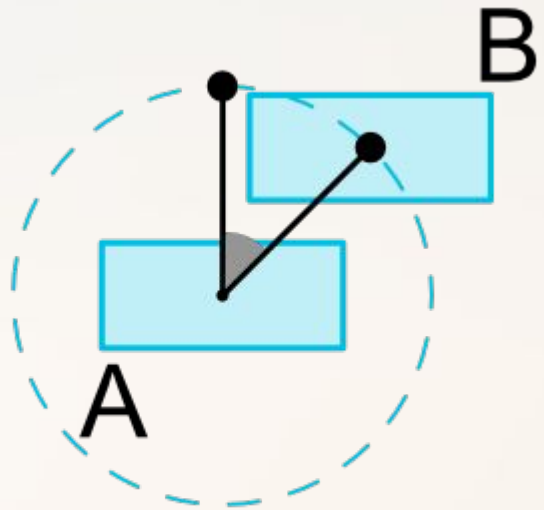


Cada restricción es una conexión/alineación con otra vista, el Layout padre, o una guía invisible. Si se suelta un view en un constraintLayout y no se añaden restricciones, se queda donde está en el editor, pero se dibujará en (0,0). En este ejemplo, a C le falta una restricción vertical, por lo que se alineará al top.

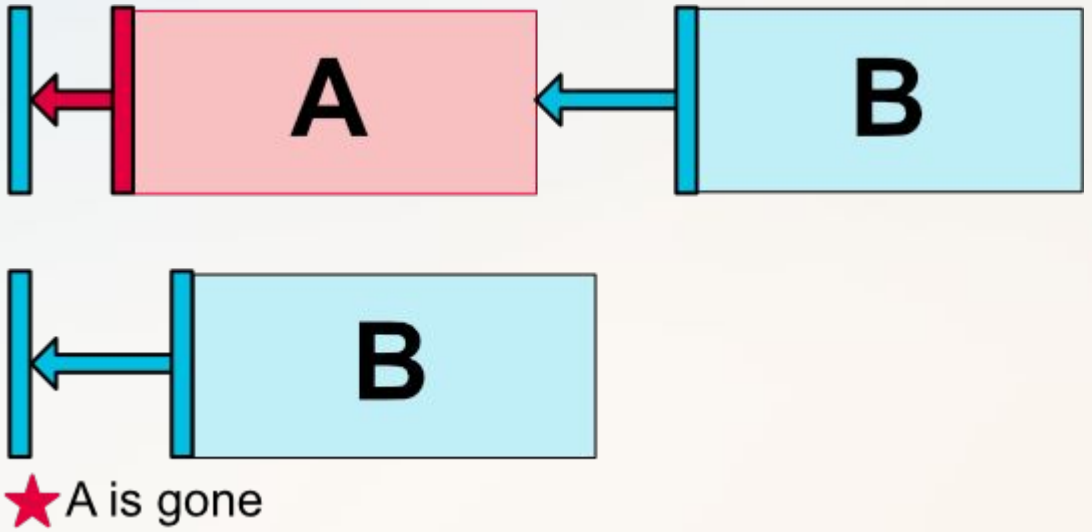
Se pueden disponer los elementos de diferentes formas con distintas restricciones (que son los parámetros de diseño de ConstraintLayout). La forma más sencilla es mediante Posicionamiento Relativo a otro elemento o al borde.



Se pueden especificar otros tipos de posicionamiento relativo, como a partir de un radio y un ángulo, con layout_constraintCircle layout_constraintCircleRadius layout_constraintCircleAngle



Hay que tener ojo porque la visibilidad o no de algunas vistas, afecta a la forma de mostrarse de otras.

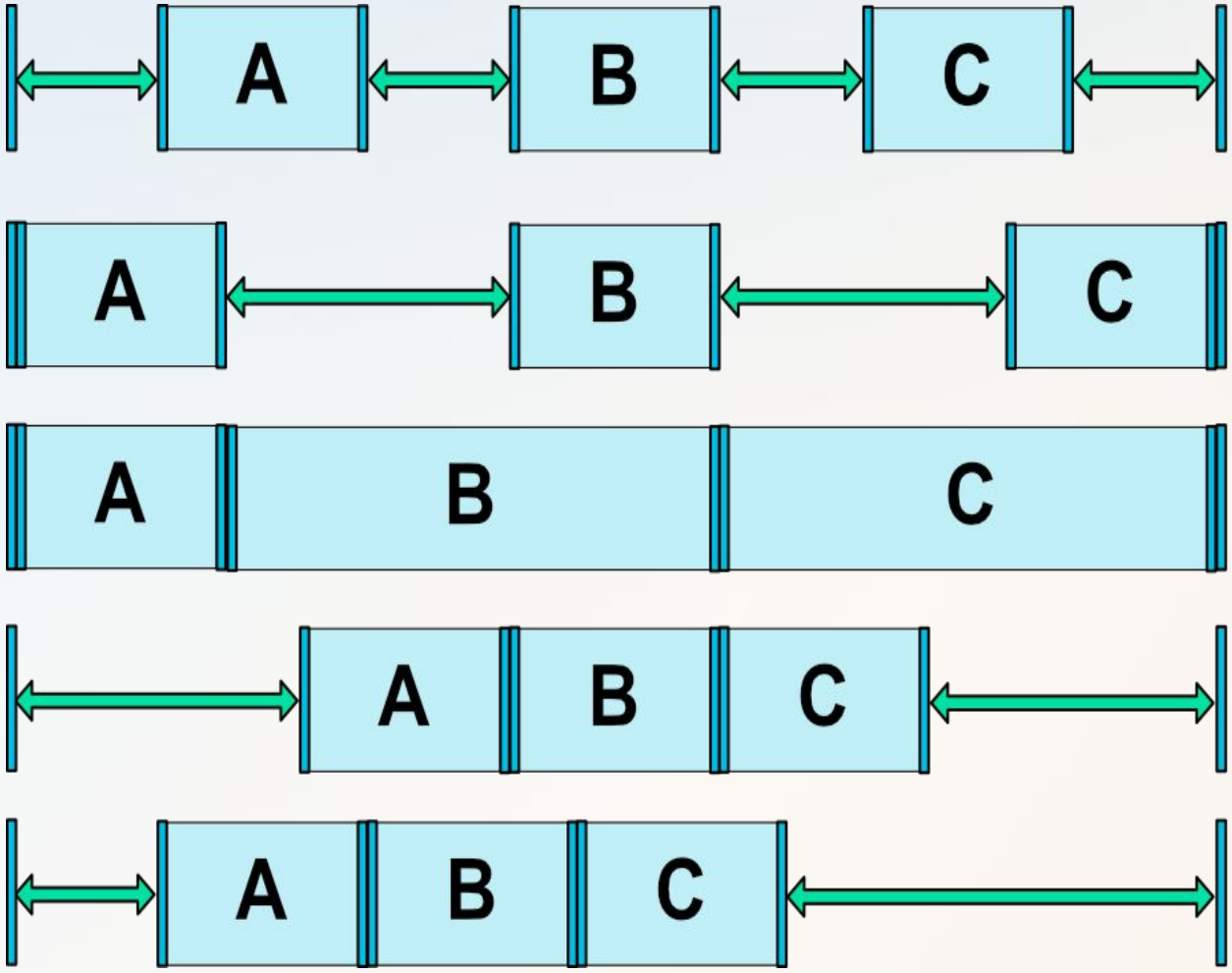


Si encadenamos restricciones correctamente, también podemos formar una cadena de vistas. Si seleccionamos varios elementos en la vista de diseño, podemos dar a click derecho -> chain -> create chain (Vertical u horizontal).

Una vez hecho esto, el elemento inicial puede determinar la forma de la cadena con otros parámetros de diseño:

layout_constraintHorizontal_chainStyle
layout_constraintVertical_chainStyle
spread, spread_inside, packed

solo spread: app:layout_constraintHorizontal_chainStyle="spread"
app:layout_constraintHorizontal_weight="0.5"



Spread Chain

Spread Inside Chain

Weighted Chain

Packed Chain

Packed Chain with Bias

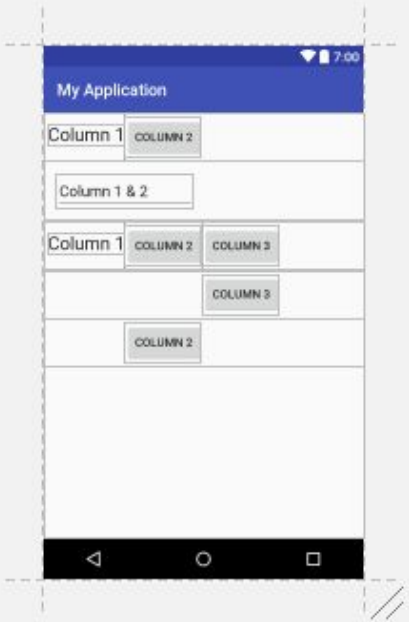
RelativeLayout es un ViewGroup que muestra a sus hijos en posiciones relativas. La posición de una vista se puede especificar relativa a la de otra o a la del padre.

La introducción de ConstraintLayout deja a este obsoleto, por lo que se recomienda NO USARLO.

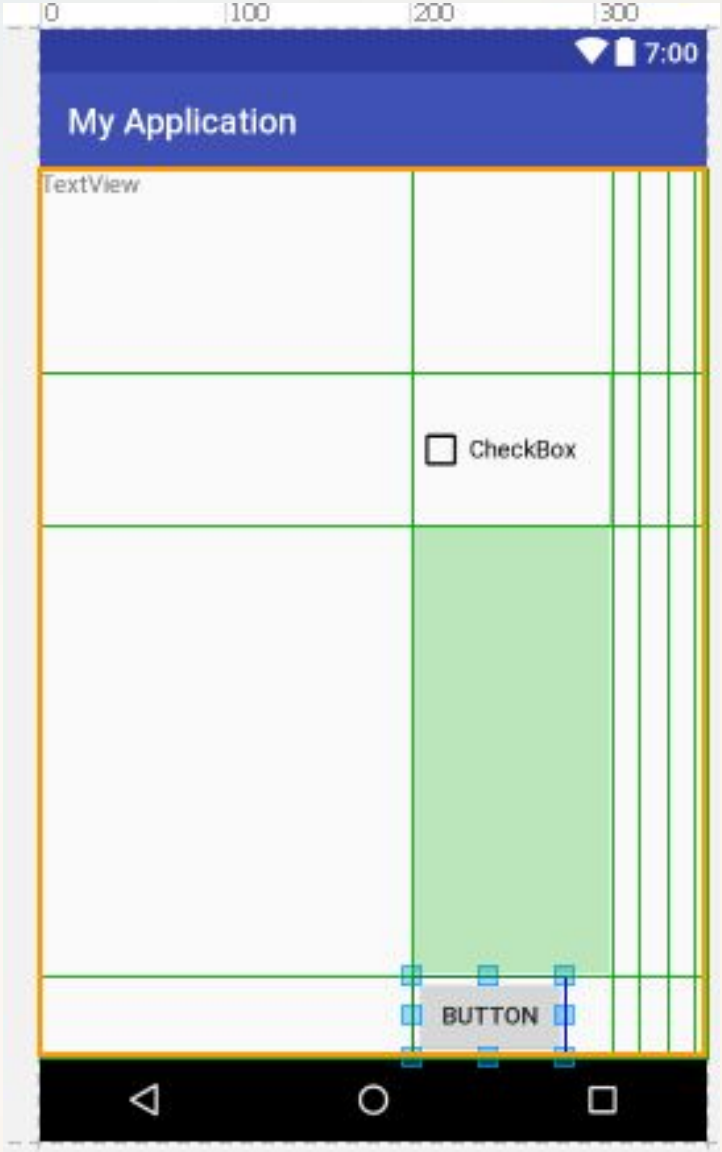
TableLayout ordena a sus view hijas en filas y columnas (sin borde). Contendrá elementos TableRow. Cada fila tiene una o más celdas (elementos). El número de columnas es el de la fila con más celdas. El ancho de cada columna viene dado por el ancho más grande de los elementos de esa columna.

Los parámetros de diseño más importantes son android:layout_column y android:layout_span. También hay otros que se aplican a la misma vista Tabla, como android:stretchcolumns

```
<!-- 2 columns -->
<TableRow
    android:id="@+id/tableRow1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="5dip" >

    <TextView
        android:id="@+id/textView1"
        android:text="Column 1"
        android:textAppearance="?android:attr/

    <Button
        android:id="@+id/button1"
        android:text="Column 2" />
</TableRow>

<!-- edittext span 2 column -->
<TableRow
    android:id="@+id/tableRow2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="5dip" >
```



GridLayout muestra a sus vistas hijo en una rejilla imaginaria, determinada como una serie de filas y columnas, cuyo ancho y alto viene determinado de la misma forma que TableLayout. De hecho, es una evolución suya. Tiene sus propios parámetros de diseño, aunque son muy parecidos: https://developer.android.com/reference/android/widget/GridLayout.LayoutParams.html



Lo normal es intentar que, a no ser que se trate de texto, no se tenga que hacer scroll nunca en una pantalla de Android. Pero si fuese imprescindible, tenemos disponibles ScrollView y HorizontalScrollView. No son Layouts, pero son viewGroups: Permiten contener elementos dentro. Si contienen un LinearLayout o un ConstraintLayout, (o cualquier otro layout) estos mostrarán Scroll bars si es necesario, en la dirección indicada. Si necesitas scrollbars en ambas direcciones, es posible hacerlo con el atributo scrollbarAlwaysDrawVerticalTrack o scrollbarAlwaysDrawHorizontalTrack. Pero si necesitas ambos, algo estás diseñando mal.

5 - Atributos

Imagen de Giphy.com

Los Atributos en los layouts de Android se escriben como los atributos en HTML o XML (porque lo son de XML). Sirven para modificar las propiedades o el comportamiento de los layouts y los elementos. Cada uno de ellos tiene su apartado en el editor visual, y un atributo asociado a la etiqueta XML. la mayoría de veces comienzan con android:.

No todos los atributos tienen sentido con todos los elementos. Vamos a ver los más interesantes, pero no todos. Tenéis una lista completa en la documentación oficial: https://developer.android.com/reference/android/view/View

- id: Sirve como identificador, y lo podemos usar para asociar eventos al elemento. La @ indica que el analizador de XML debe identificar el String como una Id. El + significa que es un nuevo nombre de recurso que se debe crear y agregar a nuestros recursos (R)
  android:id="@+id/miId"

- alpha: Recibe un valor entre 0 y 1, indicando la opacidad. Por defecto 1.
  android:alpha=".5"
- background: Puede recibir o un recurso drawable (como una imagen) o un color en forma hexadecimal…
  Podemos usarlo con un recurso en drawable. En ese caso se usa referenciando a @carpeta/imagen/nombreimagensinextension
  android:background="@drawable/sputnikfondo"
  Lo mismo para referenciar un mipmap:
  android:background="@mipmap/asdf_round"
  Se pueden indicar colores así: #aarrggbb

CENEC MALAGA

¡Ojo con los botones!

En las nuevas versiones de Android Studio en 2019, los botones tienen por defecto un fondo con reborde blanco, para simular botones con bordes redondeados por defecto. Hace que parezca que hay un margin en todos. Para deshacernos de él, simplemente hay que cambiar el color de fondo con el atributo background.

¡Además!

El tema Material no permite cambiar fondo de botones con android:background. Hay que usar android:backgroundTint

- backgroundTint: Recibe un color hexadecimal, e indica cómo se dibujará la parte no transparente del fondo:
  android:backgroundTint="#ff0000"

- clickable: Establece si reacciona a los click o no. Tiene un valor true o false
  android:clickable="true"

- focusedByDefault: Recibe true o false, e indica si este elemento tiene el foco por defecto. Solo se puede usar en un elemento por pantalla. Tiene sentido en los campos de texto.
  android:focusedByDefault="true"

- foreground: Color o imagen (o recurso) que se dibuja por delante del background, y todos los elementos que contiene si es que es un layout.
  android:foreground="#ff0000"

- foregroundTint: Análogo a BackgroundTint.

- keepScreenOn: Recibe true o false, e indica si la presencia de este elemento debería mantener la pantalla encendida mientras esté visible
  android:keepScreenOn="false"

- layoutDirection: Solo se aplica a los layouts, indican la dirección de dibujado. Se puede usar para adaptar a idiomas que escriben de derecha a izquierda, como el árabe. Sus valores principales son ltr y rtl
  android:layoutDirection="rtl"

- minHeight / minWidth: Especifican dimensiones mínimas. No se asegura que se cumplan, las restricciones de posicionamiento pueden más.
  android:minHeight=”20dp"

- onClick: De los más importantes. Establece qué función en la actividad (Lógica detrás del diseño) manejará el evento click de esta vista.
  android:onClick="miMetodo"
  Esto tiene su reflejo en una función dentro de la actividad:

```
public void miMetodo(View view) {
    ...
}
```

- padding: La separación interna entre los bordes del elemento y su contenido:
  android:padding=”10dp”
  Tenemos todas las variantes de HTML: paddingTop, paddingLeft…

- rotation: Expresa la rotación del elemento en grados, con un parámetro flotante:
  android:rotation=”14.5”
  Tiene los análogos rotationX y rotationY, que rotan los elementos solo de un eje.

- scaleX / scaleY: Admite un float que indica la escala que tendrá el elemento respecto a su dibujado normal
  android:scaleX=”1.5”

- textAlignment: Útil para los tags de texto, indica dónde se alineará el texto.
  android:textAlignment=”center”
  Puede ser textStart, textEnd, viewStart, viewEnd, o center. La diferencia entre los textXXXX y los viewXXXX es que se comportan de forma distinta según la orientación.

- textDirection: Análogo a LayoutDirection, para textos.

- tooltipText: Establece un texto que se mostrará como ayuda cuando se haga una pulsación larga sobre el elemento.
  android:tooltipText=”texto de ayuda”

- hint: Sirve solo para los campos de texto, recibe un texto que se usa como placeHolder de ese campo.

- visibility: Establece la visibilidad del elemento. Este es también muy importante. Tres valores posibles: visible, invisible y gone.
  android:visibility=”gone”
  Visible quiere decir que es visible
  Invisible no muestra la vista, pero ocupa su espacio.
  Gone la oculta y además hace que no ocupe espacio.

- gravity: Especifica cómo se debería posicionar el contenido de un elemento en los ejes X e Y. Sus valores posibles son: top, bottom, left, right, center, center_vertical, center_horizontal... No tiene mucho sentido en un ConstraintLayout, que dibuja los elementos de otras formas, pero si en un LinearLayout:

```
android:gravity="center_vertical"
```

- layout_weight: Indica en un LinearLayout qué proporción de tamaño tendrá un elemento sobre otros. Vamos a hacer experimentos, poniendo un peso de 1 a varios elementos, luego uno de 2 a uno de ellos…

- margin: tiene variantes margin_horizontal, margin_vertical, margin_top, etc… establece un margen como el de HTML y CSS

- textSize: Indica el tamaño del texto ¡ACUÉRDATE DE USAR SIEMPRE UNIDADES SP!

- textStyle: permite elegir estilos de texto entre bold, italic o normal. Se pueden combinar con el operador | . Ej: bold|italic aplica ambos efectos

- layout_constraintWidth_percent recibe de valor un numero entre 0 Y 1 y da la proporción que ocupa el elemento dentro de un constraintLayout. El elemento debería tener el tamaño a 0dp. Equivalente a esta es layout_constraintHeight_percent para la altura. ej:

```
app:layout_constraintWidth_percent=".70",
app:layout_constraintHeight_percent=".30"
```

6 - Nuestros primeros experimentos en Java

CENEC MALAGA

Imagen de Giphy.com

Ya que hemos tocado los Layouts, vamos a echar un vistazo a la clase de la actividad principal en Java:

```
public class MainActivity extends
AppCompatActivity {

   @Override
   protected void onCreate(Bundle
savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
}
```

- Que extienda a Activity o a AppCompatActivity, nos indica que se corresponde con una Actividad, es decir, una “Pantalla” de nuestra app.

- El método onCreate es el primero que se ejecuta en el ciclo de vida de la App. Ahondaremos en él dentro de poco.

- super.onCreate llama al método onCreate de la clase padre. Es muy recomendable hacer esto con todas las funciones que sobreescriban un paso del ciclo de vida de Android.

- setContentView nos relaciona esta actividad con su Layout. La clase R representa la carpeta de recursos “res”. Dentro de ella, buscamos en Layouts, y seleccionamos el layout específico para esta Actividad. Recuerda recompilar si añades un nuevo Layout para que Java lo detecte.

Deberíamos poner una id a cada vista de nuestro Layout, porque así, seremos capaces de traernoslo a Java, seleccionando R.id.miId.

Tenemos además, una función que podemos usar por heredar de Activity o AppCompatActivity, que es:

this.findViewById(R.id.***miId***)

Esto nos devuelve un objeto de tipo T. Quiere decir que es un parámetro. Nos puede devolver varios tipos de dato.

Tenemos disponible en Java/Kotlin una clase por cada tipo de vista (TextView, Button, Switch…) que podemos usar en el Layout, con el mismo nombre. Podemos igualar el resultado de findViewById a cualquier tipo. Tenemos que hacerlo, al tipo que tengamos declarado en el Layout para esa id.

Por ejemplo, si en nuestro Layout tenemos el elemento:

```
<TextView
   android:id="@+id/miId"
   android: …
```

En Java podemos invocar ese elemento usando:

```
TextView etiqueta1=this.findViewById(R.id.miId);
```

¡Vamos a experimentar con las funciones que nos permite TextView! Cada vista tiene sus propias funciones. Mira la documentación de cada una para saber utilizarlas, y usa el sentido común: Probablemente haya una función para cada uno de los usos típicos de cada elemento.

Es el momento de utilizar onClick, el atributo que aprendimos en el punto anterior. Vamos a probar en clase distintos elementos que interaccionen entre ellos.

Podemos crear el método onclick de dos formas:

- A través del atributo android:onClick del Layout, y creando la función sugerida en el código Java:

- Obteniendo el elemento desde Java, y utilizando un onClickListener:

```
miBoton.setOnClickListener(new
View.OnClickListener() {
        @Override
        public void onClick(View v) {
          //Cosas
        }
    });
```

También tenemos disponible a nuestro amigo el Mensaje Toast, que es un mensaje emergente en el dispositivo, que se lanza aunque la aplicación esté en segundo plano. Se lanza así:

```
Toast.makeText(getActivity() , “Hola, soy un toast”, Toast.LENGTH_LONG).show();
```

El primer argumento es un contexto (ya veremos lo que son), el segundo un mensaje o recurso en R (podría estar en Strings.xml), y el tercero una longitud, indicada por una de varias constantes disponibles. Documentación completa: https://developer.android.com/reference/android/widget/Toast

¡Cuidado con Toast!

Si imprimes varios Toast seguidos, en las últimas versiones de Android **(Desde la 8.0 hasta Android 10.0)**, se muestran todos los Toast a la vez, uno encima de otro. Por lo que solo verás el último.

En versiones más antiguas, y desde la 10 en adelante de nuevo, se muestra uno tras otro, se acaba de mostrar uno antes de mostrar el siguiente.

Esto se podría usar en nuestra ventaja para depurar ;)

Podemos añadir imágenes a nuestra app. Si son iconos de aplicación o de notificación, deberían ir a res/mipmap. Se añaden Haciendo click derecho en la carpeta -> Nuevo -> Image Asset. A partir de ahí, se puede seguir una pantalla de configuración bastante intuitiva. El icono de la app se puede cambiar desde el XML del Manifest. ¡Busca dónde! No te debería ser difícil encontrarlo.

Para añadir imágenes dentro del Layout de nuestra aplicación, primero tenemos que tenerlas en la carpeta drawable. Haz click derecho en la carpeta, selecciona “Show in Explorer”, y mete tu recurso ahí. Recuerda recompilar para que esté disponible. MUY IMPORTANTE: NO DEBEN TENER MAYÚSCULAS, ESPACIOS, NI CARACTERES ESPECIALES EN EL NOMBRE.

Una vez allí, podemos desde el editor de diseño de Android, añadir un nuevo ImageView. Cuando lo soltemos, se nos abrirá un diálogo, desde el que podremos seleccionar el recurso.

Una vez insertado, se nos quedará en el xml del Layout algo como esto:

```
<ImageView
   android:id="@+id/imageView2"
   android:layout_width="276dp"
   android:layout_height="161dp"
   app:srcCompat="@drawable/miImagen"
   ...
```

Con los distintos valores del atributo android:scaleType puedes ajustar las distintas formas de visualización y ajuste de la imagen.

También puedes poner una imagen de fondo a un ViewGroup con el atributo android:background:

```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   ...
   android:background="@drawable/miImagen"
...
```

Si queremos un escalado distinto al que hay por defecto, lo más sencillo es usar un FrameLayout como padre, que contenga una imagen a tamaño completo, y encima, el ViewGroup que queramos con el contenido. Ya aprenderemos más formas.